NCR

INVENTION DISCLOSURE RECORD

LAW DEPARTMENT USE ONLY	
Docket No.	9362
Date Received	
Attorney	

PREPARATION & ROUTING INSTRUCTIONS

Complete and fill in every item. Writ "none" or "unknown", if appropriat .

Use an additional blank page for any item where more space is needed.

Have your manager review and sign (items 9 and 10) before submitting to the NCR Law Department.

Submit original and one copy to: NCR Corporation, Intellectual Property Section, Law Department, ECD-2, 101 W. Schantz Avenue, Dayton, Ohio 45479. *Keep one copy for your file.*

(1) Inventor(s)	Facility	Department	Phone Number
Gregory Milby	Rancho Bernardo	9515	858-485-3447
Steve Grolemund	Rancho Bernardo	9515	858-485-3425
Susan Choo	El Segundo	9515	310-524-7957

(2) Title of Invention (Preferably 10 words or less)
Shadow Map Management of DBS Temporary and Permanent Storage

(3) Product, Project Name or Class Number	(4) Date invention was First Conceived	(5) Actual or Anticipated Date of First Product Sale, Customer Availability, or Public Disclosure
TOR	3/26/2000	2001

(6) Description of the Invention
Please attach additional pages providing the following:

a. **Statement of problem solved by the invention** - Briefly state the problems your invention solves, its purposes and advantages, and how it differs from prior designs that you are aware of.
b. **Description of the invention** - Describe your invention in detail. Include and refer to sketches or diagrams and, if appropriate, attach documents such as previously prepared descriptions or specifications.
c. **Summary of Invention** - State what you regard at the present as the key inventive concept - i.e., the gist of your invention.

(7) Inventor Signature(s) (Each person listed in Item 1 above is an inventor and must sign and date.)

Signature of Inventor	Date	Signature of Inventor	Date
Signature of Inventor	Date	Signature of Inventor	Date

(8) Witness Signatures (Two persons who are not inventors must read and understand this disclosure, and then sign and date.)

Signature of Witness	Date	Signature of Witness	Date

FOR MANAGER USE ONLY

(9) Strategic Value of Patent Coverage (State what you regard as the strategic value to your business unit of having a patent for this invention - e.g., licensing revenue, preventing use by others, importance/breadth of the invention, etc.)

(10) Review d and approv d by

Signature of Manager	Date	Manager Name (Please print)	Tentative Rating * (A, B, C, D, or U)

* Ratings of "A" through "D" indicate relative value, with "A" being highest and "D" being lowest. A rating of "U" indicates the value is unknown.

Probl m Solved By Invention

Modern database systems divide storage usage into system, permanent, and temporary data storage regions. Ideally, the same storage pool can be used to supply storage to satisfy all of these needs. Of the three, the two primary consumers of storage are temporary and permanent storage. User data is stored in Permanent storage, whereas Temporary Disk Storage houses th intermediate or final results of a client's query. Although both forms of storage contain, at times, user data, the behavior of the database logic, on behalf of the two types of storage, varies dramatically with respect to: 1) need to acquire Transaction Locks, 2) logging of data changes within the storage region, and, 3) levels of data consistency achieved in the event that a data recovery cycle is required. More specifically, it is expected that permanent file management context be consistent across system crashes or resets, whereas it is expected that temporary file management context be essentially erased in the event of a system crash or reset. A further motivation for segregating temporary and permanent file behavior is performance. A substantial performance gain can be achieved by allowing temporary file management operations to by-pass costly transaction lock, and database logging operations. The problem to be solved, and which is solved by this invention, is to enable both Permanent and Temporary Storage to be supplied by the same storage pool while at the same time providing an environment which enables the database software to treat the two storage types so dramatically different.

The invention provides three basic features: 1) providing the user with an application interface (API) and supportive logic that will enable them to manage both Permanent and Temporary storage from a common pool. 2) Providing logic to manage Temporary storage without requiring the acquisition of any transaction locks nor provoking any database logging activity, whereas the management of Permanent storage will be accompanied by a behavior which does result in both the acquisition of transaction locks, as well as provoking database logging activity. 3) Permanent File Management context will be consistent across system resets or crashes, whereas Temporary File Management context will basically be "erased".

Description of Invention

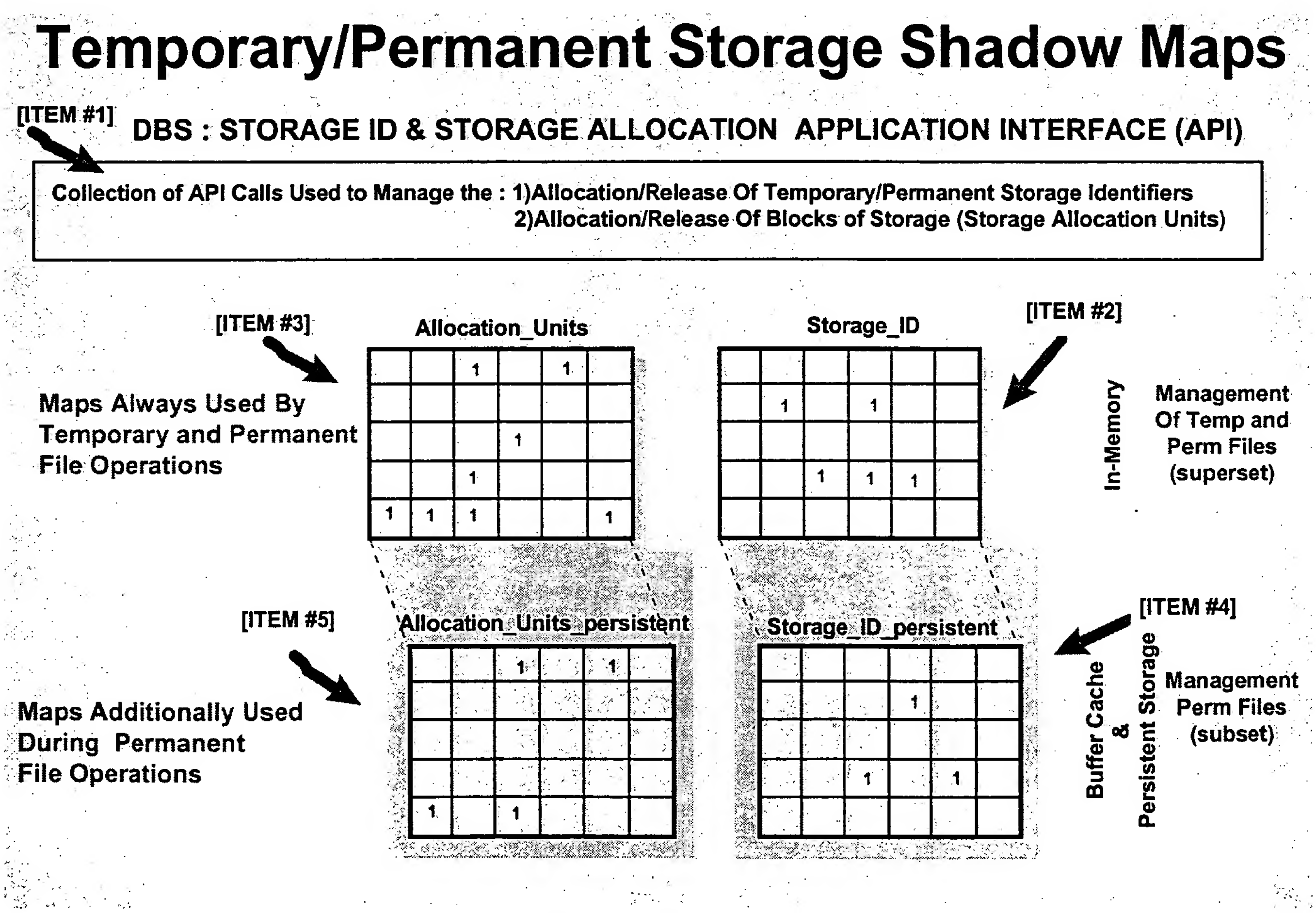

Figure 1 : In-Memory and Persistent (shadow) maps used to manage Permanent/Temporary Storage

The above diagram depicts the major components involved in the invention. In the diagram shown, a single application interface (API) is used to manage both permanent and temporary storage. This API provides management of both permanent (including system) and temporary storage which is being supplied from a common pool. The distinction between permanent and temporary space is accomplished by passing a flag

parameter; identifying the class of storage desired, with each storage management call. Amongst the services that the API must provide, are the service routines which:
1. Assign a unique identifier to identify each permanent or temporary storage file to be created.
2. Assign or Free blocks of storage (storage allocation units) to each of the temporary or permanent files.

These two services are implemented by creating two maps. One map keeps track of storage IDs which have already been allocated [ITEM #2] (and thus are in use), and another map keeps track of storage allocation units (contiguous series of blocks of storage) which have been allocated/assigned to the various permanent and temporary files [ITEM #3]. A common Application Interface (API) is used [ITEM #1] to access, search, and manipulate the maps. In the diagram, an allocated Storage Identifier, or allocated storage allocation unit, is depicted by flipping a flag bit to "1". The Storage ID [ITEM#2] and Allocation Unit [ITEM #3] maps are maintained solely within volatile main memory. The logic for the service routines ensures that writing to either the in-memory Storage ID or Allocation Unit map does not result in: 1) The acquisition of any Transaction Locks, 2) The writing of any database log records. The reason for this will soon become clear.

Each of the two principal maps are "shadowed": the storage ID map [ITEM #2] is shadowed by the corresponding persistent version of this map [ITEM #4]; the Allocation Unit map [ITEM #3] is shadowed by its corresponding persistent version [ITEM #5]. The shadow maps are persistent in that they are maintained in storage and read into memory, modified, then written back out to storage, whereas as the principal maps are maintained solely in memory. The logic which manipulates these maps ensures that any changes to the persistent versions of these maps are subject to: 1) Transaction Locking, 2) Logging of data changes made to the maps, and, 3) Guarantees of data consistency across restart boundaries. The reasons for this behavior will soon be made apparent.

At start-of-system, the persistent versions of the Storage ID [ITEM#4] and Allocation [ITEM#5] are read into memory. They are then copied in order to create the volatile in-memory versions of the Storage ID [ITEM#2] and Allocation [ITEM#3] maps. Thus, at start-of-system, each volatile in-memory map, and its corresponding persistent shadow, are equal.

Allocation of a storage identifier for a Temporary File is accomplished by first searching the in-memory Storage ID map [ITEM #2] for a free ID, and then setting a bit in the volatile in-memory Storage ID map [ITEM #2]. Allocation and assignment of units of storage to the new temporary file identified by storage ID is accomplished by first searching the in-memory Allocation map [ITEM#3] for free storage units, and setting the corresponding bits in the volatile in-memory Allocation Map [ITEM #3]. The setting of bits in the in-memory Storage ID [ITEM #2] and in-memory Allocation Map [ITEM #3] does not result in any transaction lock, or logging activity. Since these entries are made only in the volatile versions of the maps, the changes will be lost in the event of a system crash or reset. Upon completion of the reset cycle the persistent versions of the Storage ID [ITEM#4] and Allocation [ITEM#5] will be read into memory and then copied to form the volatile in-memory Storage ID [ITEM#2] and Allocation [ITEM#3] versions. This action has the effect of "returning" all of the temporary storage space that was in-usage prior to the reset, back to the system.

Allocation of a storage identifier for a Permanent File is accomplished by first searching the in-memory Storage ID map [ITEM #2] for a free ID, and then setting both a bit in the volatile in-memory Storage ID map [ITEM #2], as well as in the persistent Storage ID shadow map [ITEM #4]. Allocation and assignment of units of storage to the new permanent file identified by the storage ID is accomplished by first searching the in-memory Allocation map [ITEM#3] for free storage units, and then setting the corresponding bits in both the in-memory Allocation Map [ITEM #3] as well as the persistent Allocation shadow map [ITEM #5]. The logic that performs the writing to the persistent Storage ID [ITEM #4] and Allocation [ITEM #5] maps will also perform the necessary transaction lock and logging activity. Entries made in the persistent versions of the maps will also possess the property of being recoverable across a system crash or reset.

From the above discussion it can be deduced that the in-memory versions of the Storage ID [ITEM#2] and Allocation [ITEM#3] are a superset of their respective persistent versions, Storage ID[ITEM#4] and Allocation [ITEM#5]. It can be further deduced, that:
1. The in-memory maps represent the union of file management context for the temporary files and permanent files.
2. The persistent version of the maps contains the file management context for the permanent files.
3. The portion of the in-memory maps that are not contained in their corresponding persistent versions contains the file management context for the temporary files.

The requirement that the search for free IDs and free allocation units utilize the in-memory versions of the Storage ID [ITEM #2] and Allocation [ITEM #3] maps is what provides the user with the ability to allocate both permanent and temporary storage from a common pool.

Summary of Inv ntion

The invention "Shadow Map Management of DBS Temporary and Permanent Storage" provides the DBS designer with an approach for managing both Permanent and Temporary Storage in a seamless manner. The key to the design is the "shadowing" of the primary usage in-memory maps, Storage ID [ITEM #2] and Allocation [ITEM #3] maps, with their respective persistent counterparts, Storage ID [ITEM#4] and Allocation [ITEM #5].

With regards to the first claim of the invention: The in-memory versions of the maps are supersets of their corresponding persistent counterparts and contain the file management context for both permanent and temporary files. The persistent versions of the maps ("shadowed" versions) house only a copy of the permanent file management context. The claimed feature of providing the DBS designer with the ability to manage both temporary and permanent storage from the same pool, is being accomplished by the fact that the API logic utilizes exclusively the in-memory version of the maps when searching for free storage ID's and free storage allocation units, and the logic always sets bits in this in-memory version regardless of the type of storage being allocated.

With regards to the second claim of the invention: The behavior of the logic when flipping bits in the in-memory versions of the maps is orthogonal to the behavior which occurs when flipping bits in their persistent counterparts. Only the in-memory versions of the maps are used when assigning storage IDs and/or storage allocation units on behalf of a temporary file. The flipping of the bits within the in-memory maps results in no acquisition of any transaction locks and no database logging activity. The assignment of storage ID's and/or storage allocation units on behalf of a permanent file requires the flipping of bits within the in-memory versions of the maps as well as the flipping of the same bits within the persistent map counterparts. The flipping of bits within the persistent versions of the maps results in the acquisition of the appropriate transaction locks and in database logging activity. It is this difference in behavior, between flipping bits in the in-memory versions of the map and flipping bits in the persistent versions of the map, which leads to the invention claim that an environment has been provided which allows both permanent and temporary storage to be managed from a common pool while also allowing the orthogonal behavior, with respect to whether or not transaction locking and database logging occurs while managing the two distinct storage classes.

The third invention claim, that the permanent file management context be consistent across database system crashes or resets, whereas the temporary file management context be vanquished upon a system crash or reset, is accomplished by housing the both the temporary and permanent file management context in the volatile in-memory versions of the maps, and housing a copy of only the permanent file management context with the persistent versions of the maps. Upon a system reset or system crash, the volatile versions of the maps disappear, leaving only the persistent versions of the maps. Thus the temporary file management context is simply, "erased".